Department of Electrical and Computer Engineering
University of Thessaly

Master Thesis

Cognitive Radio Systems

By
Charalampos Manolidis

Volos
February 2018

UNIVERSITY OF THESSALY
DEPARTEMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Cognitive Radio Systems

(Υλοποίηση Αλγορίθμων Δυναμικής Προσαρμογής Ευφυών Συστημάτων Επικοινωνίας)

Master Thesis

Charalampos Manolidis

Supervising Professors: Korakis Athanasios
Assistant Professor

Argyriou Antonios
Assistant Professor

Katsaros Dimitrios
Assistant Professor

Volos
February 2018

Acknowledgements

At this point, I would like to thank all those who have contributed to the completion of this work. First of all, I would like to thank my supervisor Athanasios Korakis who inspired me and gave me the opportunity to work in such an interesting project. I am grateful to PhD candidate Fraida Fund for her valuable knowledge and the help throughout the project. Also, I would like to thank the NITlab for the collaboration and the provided guidance during the project. Finally, I would like to thank my family and my friends for their support over the years.

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω όλους εκείνους που συνέβαλαν στην ολοκλήρωση της εκπόνησης αυτής της εργασίας. Αρχικά θα ήθελα να ευχαριστήσω το επιβλέποντα καθηγητή μου Αθανάσιο Κοράκη που με ενέπνευσε και μου έδωσε την ευκαιρία να ασχοληθώ με μια πολύ ενδιαφέρουσα διπλωματική εργασία. Θα ήθελα να ευχαριστήσω θερμά την υποψήφια διδάκτωρ κα Fraida Fund για την πολύτιμες γνώσεις και την βοήθεια που μου παρείχε καθ' όλη την διάρκεια της εργασίας. Επίσης θα ήθελα να ευχαριστήσω το NITlab για την αρίστη συνεργασία και καθοδήγηση που μου προσέφεραν. Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου και τους φίλους μου για την στήριξη της όλα αυτά τα χρονιά.

Abstract

In this thesis is described a set of wireless mesh experiments that performed, with instructions how to run them on various wireless testbeds. The aim of the thesis is to make those experiments and their results reproducible for the given testbeds and allow researchers with interest in wireless mesh networks to try run their own version for those experiments. For the experiments, we use indoor and outdoor wireless testbeds focusing on their wireless nodes with *IEEE 802.11* wireless cards. The first experiment introducing *B.A.T.M.A.N. advanced* routing protocol and its routing algorithm capabilities and examine the behaviour of network links when they experience failures. On the second experiment, an arbitrary network topology scenario is created and its characteristics similarly to a part of a live community network are examined.

Περίληψη

Στην παρούσα διπλωματική εργασία παρουσιάζεται ένα σύνολο από πειράματα που πραγματοποιήθηκαν σε ασύρματα mesh δίκτυα, με οδηγίες για την εκτέλεση τους σε διάφορα testbeds. Σκοπός της εργασίας είναι να καταστήσει δυνατή την αναπαραγωγή των πειραμάτων και των αποτελεσμάτων τους, δίνοντας την δυνατότητα σε οποιονδήποτε ενδιαφέρεται να πραγματοποιήσει έρευνα σε ασύρματα mesh δίκτυα, να τρέξει την δική του εκδοχή για τα πειράματα αυτά. Τα πειράματα, πραγματοποιούνται σε indoor και outdoor ασύρματα testbeds, και χρησιμοποιούνται ασύρματοι κόμβοι με ασύρματες κάρτες *IEEE 802.11*. Στο πρώτο πείραμα γίνεται μια εισαγωγή στο πρωτόκολλο δρομολόγησης *B.A.T.M.A.N. advanced* και τις δυνατότητες του αλγορίθμου δρομολόγησης του, ενώ εξετάζεται η συμπεριφορά των συνδέσεων του δικτύου όταν αυτές αντιμετωπίζουν απώλειες. Στο δεύτερο πείραμα, δημιουργείται ένα τυχαίο σενάριο τοπολογίας δικτύου και εξετάζονται τα χαρακτηριστικά του δικτύου, παρόμοια με αυτά ενός τμήματος ενός δικτύου κοινότητας.

Contents

Acknowledgements 4
Abstract....5
1. Introduction....7
2. Build a multi-hop mesh network using B.A.T.M.A.N. advanced routing protocol....8
2.1 Background....8
2.2 Results....9
2.3 Run my experiment....13
2.3.1 Set up testbeds....14
2.3.2 Set up mesh network....20
2.3.3 Force use of a relay....22
2.3.4 Introducing failures in the network....23
3. Emulations creating a specific mesh network topology....29
3.1 Background....29
3.2 Results....32
3.3 Run my experiment....40
3.3.1 Set up the testbed nodes....41
3.3.2 Set up the community network topology....41
3.3.3 Set up a mesh network with B.A.T.M.A.N.....42
3.3.4 Building on this experiment: varying link attenuation....43
References....44

1. Introduction

A wireless mesh network is a *multi-path* and *multi-hop* wireless network, consisting of a set of wireless nodes acting as network routers. Wireless mesh nodes are low-range, *peer-to-peer* wireless interconnected devices, forming a wireless mesh. The most commonly used standard in wireless mesh networking is *IEEE 802.11*, known as *Wi-Fi*. Wireless mesh networks are autonomous decentralized systems. Wireless mesh nodes cooperate one another with their nearby neighbors to exchange and transfer information across the network. [1],[2]
Packets traveling across the network hopping wirelessly from one mesh node to the next one until they reach their destination. In the case that a node needs to send its packets to a node out of range, the node sends its packets to a neighbor relay node which maintains a link to the destination of the packets. The relay neighbor will forward packets to their destination. Hence, a mesh link can be direct or multi-hop while packets traveling across the network through a sequence of relay nodes. Wireless nodes choose the best path to forward their packets depending on their routing algorithm.
Some of the main characteristics of wireless mesh networks are: *Self forming / self organizing*, *Self healing*, *Self optimization*, *Multi-hop*. [3]

Experimenting with mesh networks gives the opportunity to scientists to study wireless networks similar to ordinary community networks, not depended on central management for maintenance. Network devices that are added to the mesh network, automatically communicate between themselves in a way similar how community networks do, creating this way an identical system like the Internet but without cables or wires. This kind of network concept allows scientists to create a network that can expand every time that another router is activated nearby. Also, another attribute of mesh networks is that they cannot be completely shut-down since no one controls all routers or access points and there is no central control point. In mesh networks, we can build a network whose architecture is changing dynamically. The behaviour of the network when new devices been connected or existing devices leave can be potentially examined.

Routing is also an important factor of mesh networks performance. It is often that packets of a mesh network, cross through multiple intermediate relay nodes before they reach their destination. Wireless mesh links are lossy and it is possible that nodes have significant differences in power constraint and mobility. Conventional routing protocols are not designed for lossy and unstructured networks and they don’t consider the special characteristics of mesh networks. Mesh routing protocols take those characteristics into consideration to find the right communication route. There are two kind of routing protocols used in mesh networks, *Proactive* and *Reactive* protocols. In *Proactive Routing* protocols paths are established to all the destination

nodes regardless of whether or not the routes are needed to transmit data. In *Reactive Routing* protocols, routes are established on demand and the route discovery process is initiated when the source node requires a route to a destination node. The selection of the right mesh routing protocol can differ for different mesh environments and network requirements and can have significant impact on the network behavior and performance. [1],[2],[4]

2. Build a multi-hop mesh network using B.A.T.M.A.N. advanced routing protocol

In this experiment, we explain how to use B.A.T.M.A.N. advanced, a (dynamic) routing protocol for multi-hop ad-hoc mesh networks for setting up a mesh topology. We examine how the routes of a wireless mesh network change over time as nodes enter or leave the network.

2.1 Background

BATMAN is a proactive link-state based routing protocol that aims to offer a Better Approach To Mobile Adhoc Networking. It is designed for lossy and unstructured multi-hop mesh networks. [4],[5]

BATMAN nodes don't have to calculate the whole routes for their outgoing packets, and they don't have to know the full topology of the mesh network. In BATMAN, all nodes periodically broadcasts "hello" signals, also known as originator messages (OGM), to its neighbors. Each originator messages consists of an originator address, a sending node address and a unique sequence number. When an OGM is received, the receiving node changes the sending address to its own address and re-broadcasts the message. The sequence number is used to identify which of a pair of messages is newer. With this process, each node in the network learns its own direct neighbors, but also learns about *other* nodes that are not in range through a direct link but can be reached by hopping through a neighbor. [4]

Each BATMAN node maintains an originator table which lists the addresses of all other reachable nodes in the network. An estimation metric (TQ) is used to evaluate the link transmit quality between an originator node and a destination node. Each address entry in the originator table is assigned a TQ metric. The value of TQ metric rank is defined on a scale between 0 and 255 (0 indicating no connection and 255 being

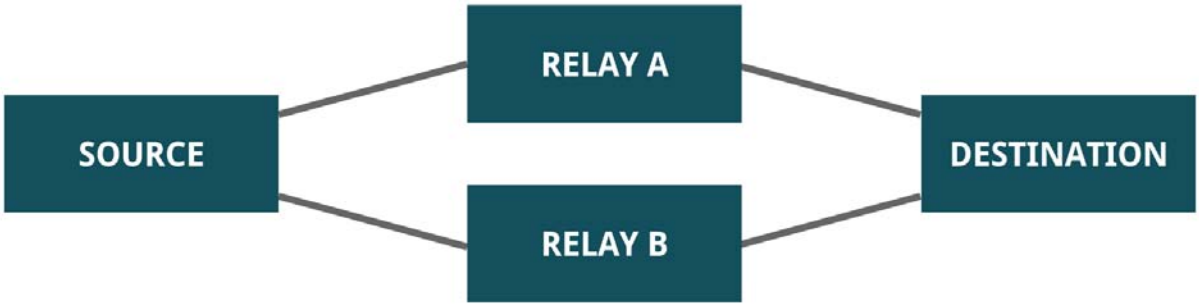
excellent link quality). When a data packet arrives for transmission, the node refers to the originator table to determine the direction in which the packets are to be sent. Specifically, it checks its originator table and forwards the packet towards the destination with the highest rank entry - the best next neighbor. [4]

When a node enters or leaves the network, or when a link fails or is degraded, this will be reflected in the TQ metric and the node/link will be avoided if a better path is available.

BATMAN nodes do not maintain the full route to every destination. Each node along the route only maintains the information about the best next neighbor through which it finds the best route. So the complete topology is not known to any single node, and topology and routing decisions are distributed among all the nodes. [4]

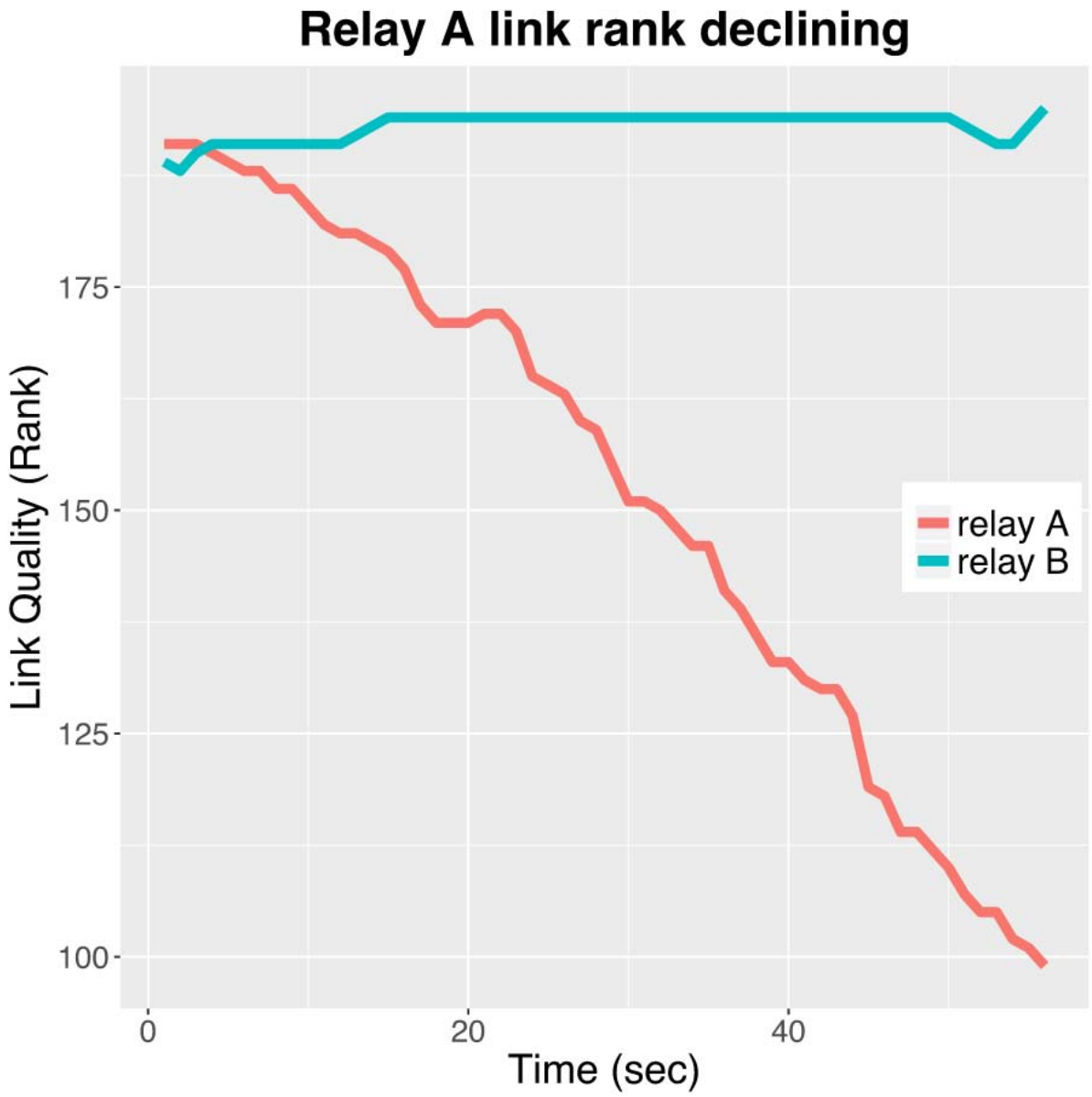
2.2 Results

In this experiment we show how link quality is estimated and how routes are formed depending on the link quality ranks (TQ). The topology that we are going to use is the following.

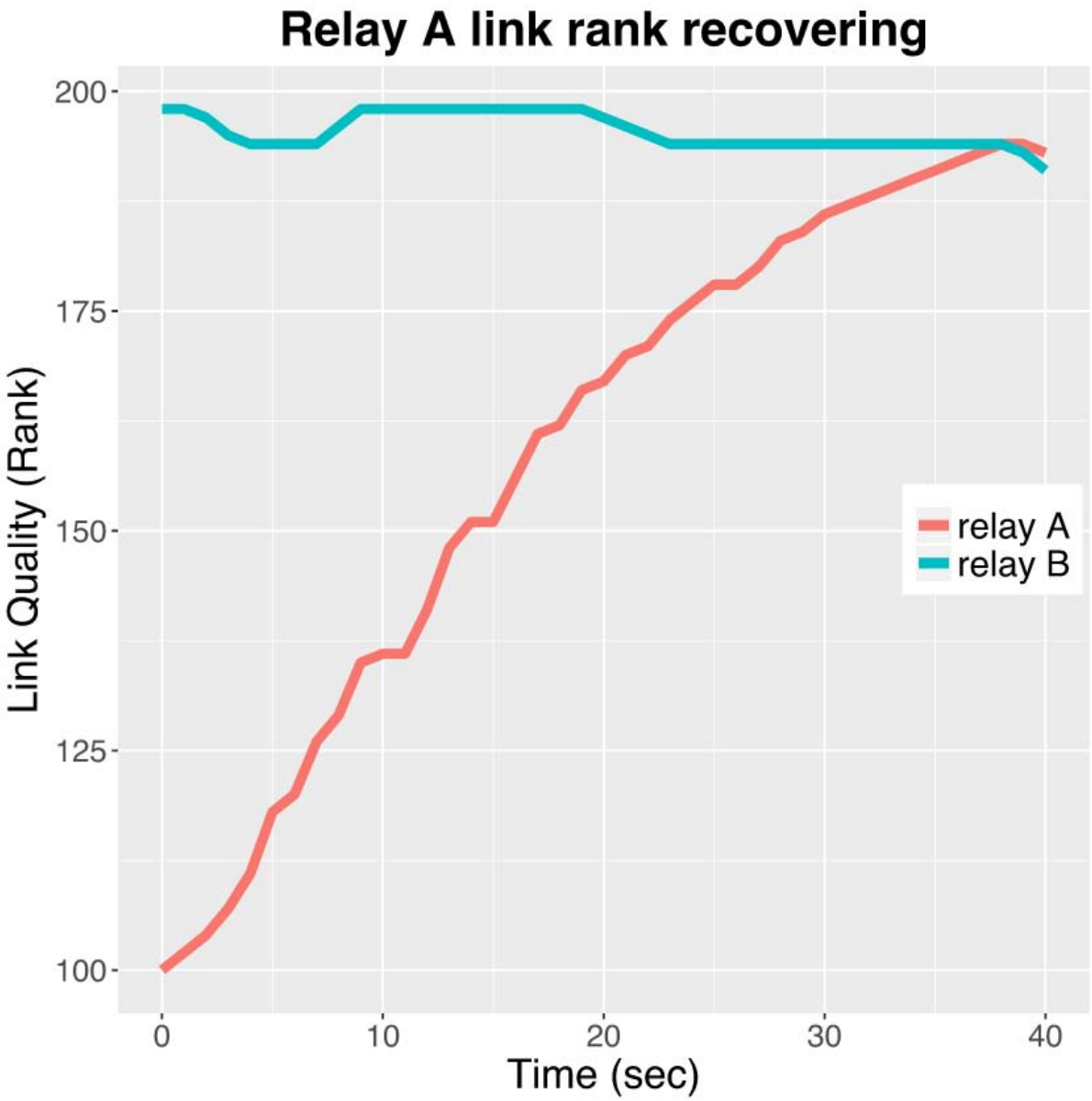


First we see how link ranks are formed and which link route is chosen at that time.

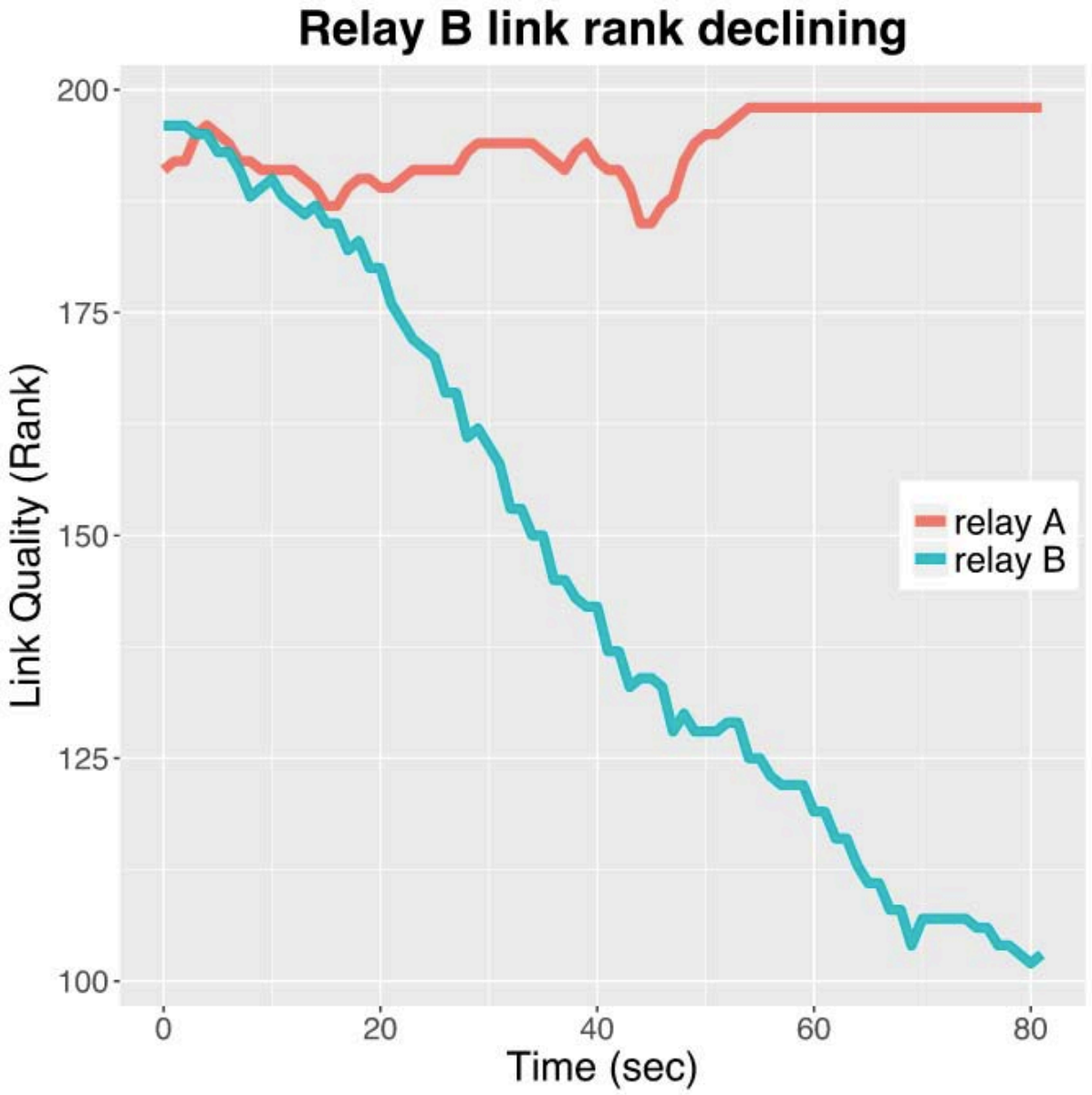
Then, we start introducing failures in the link between Relay A and the source and destination nodes, and see how the link rank drops over time and how the route is switching between the two links.



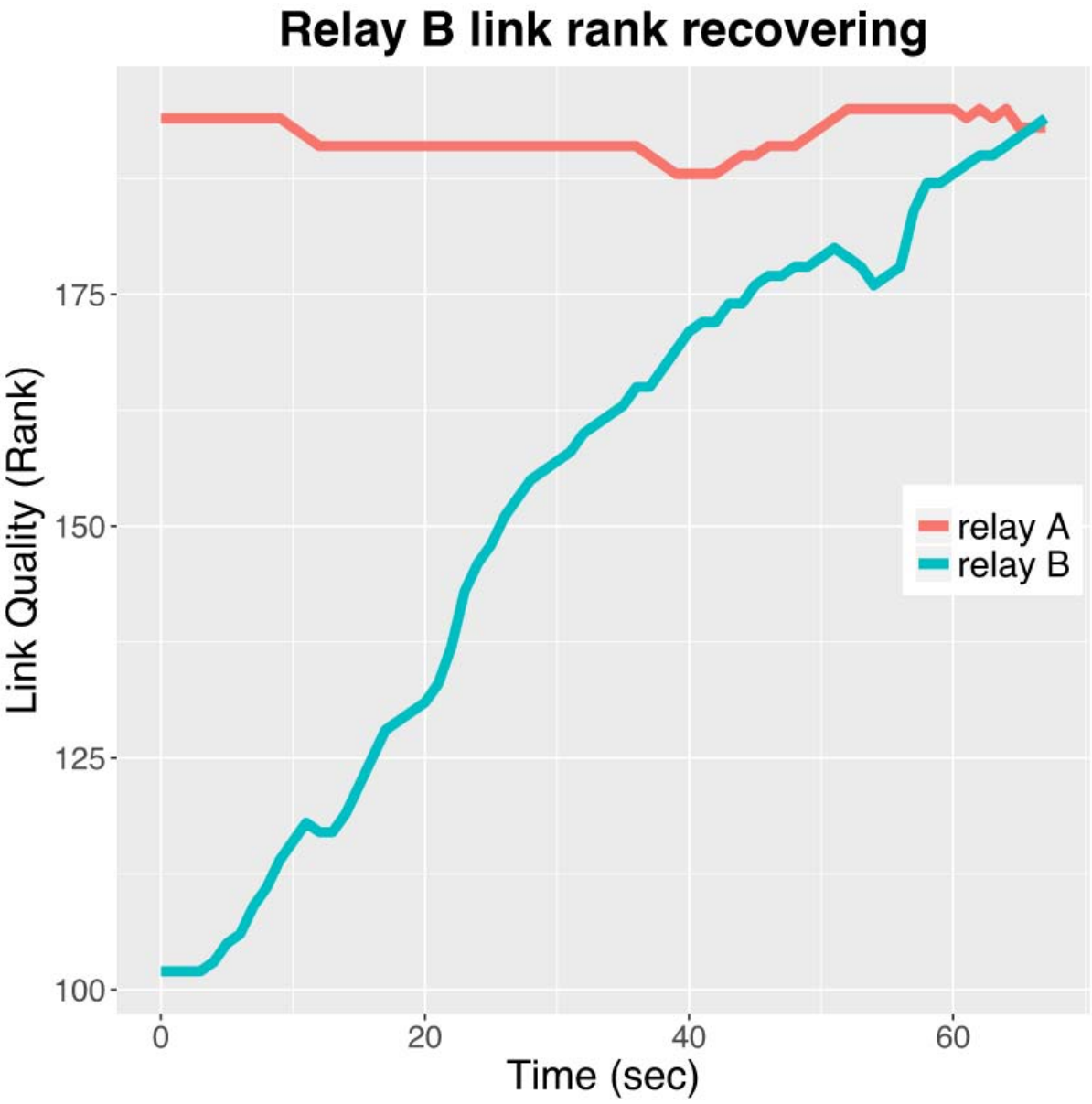
Next, we stop the failures and see how the rank of the link recovers.



When the link rank has recovered, we introduce failures to the link between Relay B and the source and destination nodes



Finally, we stop the failures again and see how the rank of Relay B recovers.



2.3 Run my experiment

In order to run this experiment, we should have a reservation on one of four GENI wireless testbeds:

- the WITest testbed,
- the "outdoor" testbed on ORBIT,
- the "sb4" testbed on ORBIT,
- or the "grid" testbed on ORBIT.

(This experiment requires four nodes with Atheros 9XXX wireless cards; these testbeds meet that requirement.)

This experiment can also run on the indoor testbed at NITOS, although you will need a separate account to use it (GENI accounts are not supported there). [6],[7],[8],[9]

2.3.1 Set up testbeds

Depending on which testbed you have reserved, you should follow the corresponding instructions to set up the testbed:

- 2.3.1.1 - Set up WITest
- 2.3.1.2 - Set up outdoor
- 2.3.1.3 - Set up sb4
- 2.3.1.4 - Set up grid
- 2.3.1.5 - Set up NITOS

2.3.1.1 Set up WITest

At your reserved time, log in to the testbed using your GENI wireless username and keys:

```
ssh -i /PATH/TO/KEY USERNAME@witestlab.poly.edu
```

Note that your GENI wireless username has a "geni-" prefix.

Identify four nodes that have Atheros 9XXX cards. In these instructions we will use node16,node17,node18,node19 if you are using WITest. But if one of these nodes is unavailable, you may replace it with another Atheros 9XXX-equipped node - any of the nodes from node16-node25.

Once we have logged in to the testbed console and identified four nodes that are available for use in this experiment, we load the `mesh-protocols.ndz` disk image onto the nodes:

```
omf-5.4 load -i mesh-protocols.ndz -t
omf.witest.node16,omf.witest.node17,omf.witest.node18,omf.witest.node19
```

(note that this is all one line, and there are no spaces around the commas).

After the disk image process has finished, we should ensure that all 4 nodes have been successfully imaged and we got a message like the following.

```
----------------------------
Imaging Process Done
4 nodes successfully imaged
----------------------------
```

If some nodes are not imaged (e.g. they do not "check in" to the experiment, or they "time out"), repeat the imaging process on just the individual nodes that failed.

Then, turn on your nodes:

```
omf tell -a on -t
omf.witest.node16,omf.witest.node17,omf.witest.node18,omf.witest.node19
```

After turning the nodes on, wait a few minutes for them to boot. Then, open *four* SSH sessions to your testbed console. In each, SSH in to another one of the four nodes as the root user, e.g. "ssh root@node16" in one session, "ssh root@node17" in the next, etc.

Then, go on to set up the network.

2.3.1.2 Set up outdoor

At your reserved time, log in to the testbed using your GENI wireless username and keys:

```
ssh -i /PATH/TO/KEY USERNAME@outdoor.orbit-lab.org
```

Note that your GENI wireless username has a "geni-" prefix.

Identify four nodes that have Atheros 9XXX cards. In these instructions we will use node1-2,node1-3,node1-4,node1-5 if you are using outdoor. But if one of these nodes is unavailable, you may replace it with another Atheros 9XXX-equipped node. To find these, visit the control panel on the ORBIT website. Click on "Status Page" and then the "outdoor" tab. Use the panels on the left side, find the "ath9k" setting under "WiFi" and check it; nodes with that hardware will be marked with an "X".

Once we have logged in to the testbed console and identified four nodes that are available for use in this experiment, we load the `mesh-protocols.ndz` disk image onto the nodes:

```
omf load -i mesh-protocols.ndz -t node1-2.outdoor.orbit-lab.org,node1-
3.outdoor.orbit-lab.org,node1-4.outdoor.orbit-lab.org,node1-5.outdoor.orbit-
lab.org
```

(note that this is all one line, and there are no spaces around the commas).

After the disk image process has finished, we should ensure that all 4 nodes have been successfully imaged and we got a message like the following.

```
----------------------------
Imaging Process Done
4 nodes successfully imaged
----------------------------
```

If some nodes are not imaged (e.g. they do not "check in" to the experiment, or they "time out"), repeat the imaging process on just the individual nodes that failed.

Then, turn on your nodes:

```
omf tell -a on -t node1-2.outdoor.orbit-lab.org,node1-3.outdoor.orbit-
lab.org,node1-4.outdoor.orbit-lab.org,node1-5.outdoor.orbit-lab.org
```

After turning the nodes on, wait a few minutes for them to boot. Then, open *four* SSH sessions to your testbed console. In each, SSH in to another one of the four nodes as the root user, e.g. "ssh root@node1-2" in one session, "ssh root@node1-3" in the next, etc.

Then, go on to set up the network.

2.3.1.3 Set up sb4

At your reserved time, log in to the testbed using your GENI wireless username and keys:

```
ssh -i /PATH/TO/KEY USERNAME@sb4.orbit-lab.org
```

Note that your GENI wireless username has a "geni-" prefix.

When you first log in to the "sb4" console, you should reset sb4's programmable attenuation matrix to zero attenuation between all pairs of nodes [7],[13]. From the "sb4" console, run

```
wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/setAll?att=0"

wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/selDevice?switch=1&port=1"
wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/selDevice?switch=2&port=1"
wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/selDevice?switch=3&port=1"
wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/selDevice?switch=4&port=1"
```

Next, identify four nodes that have Atheros 9XXX cards. In these instructions we will use node1-3,node1-4,node1-5,node1-6 if you are using sb4. But if one of these nodes is unavailable, you may replace it with another Atheros 9XXX-equipped node. To find these, visit the control panel on the ORBIT website. Click on "Status Page" and then the "sb4" tab. Use the panels on the left side, find the "ath9k" setting under "WiFi" and check it; nodes with that hardware will be marked with an "X".

Once we have logged in to the testbed console and identified four nodes that are available for use in this experiment, we load the `mesh-protocols.ndz` disk image onto the nodes:

```
omf load -i mesh-protocols.ndz -t node1-3.sb4.orbit-lab.org,node1-4.sb4.orbit-
lab.org,node1-5.sb4.orbit-lab.org,node1-6.sb4.orbit-lab.org
```

(note that this is all one line, and there are no spaces around the commas).

After the disk image process has finished, we should ensure that all 4 nodes have been successfully imaged and we got a message like the following.

```
----------------------------
Imaging Process Done
4 nodes successfully imaged
----------------------------
```

If some nodes are not imaged (e.g. they do not "check in" to the experiment, or they "time out"), repeat the imaging process on just the individual nodes that failed.

Then, turn on your nodes:

```
omf tell -a on -t node1-3.sb4.orbit-lab.org,node1-4.sb4.orbit-lab.org,node1-
5.sb4.orbit-lab.org,node1-6.sb4.orbit-lab.org
```

After turning the nodes on, wait a few minutes for them to boot. Then, open *four* SSH sessions to your testbed console. In each, SSH in to another one of the four nodes as

the root user, e.g. "ssh root@node1-3" in one session, "ssh root@node1-4" in the next, etc.

Then, go on to set up the network.

2.3.1.4 Set up grid

At your reserved time, log in to the testbed using your GENI wireless username and keys:

```
ssh -i /PATH/TO/KEY USERNAME@grid.orbit-lab.org
```

Note that your GENI wireless username has a "geni-" prefix.

Identify four nodes that have Atheros 9XXX cards. In these instructions we will use node2-7,node2-8,node2-9,node2-10 if you are using the grid. But if one of these nodes is unavailable, you may replace it with another Atheros 9XXX-equipped node. To find these, visit the control panel on the ORBIT website. Click on "Status Page" and then the "grid" tab. Use the panels on the left side, find the "ath9k" setting under "WiFi" and check it; nodes with that hardware will be marked with an "X".

Once we have logged in to the testbed console and identified four nodes that are available for use in this experiment, we load the `mesh-protocols.ndz` disk image onto the nodes:

```
omf load -i mesh-protocols.ndz -t node2-7.grid.orbit-lab.org,node2-8.grid.orbit-
lab.org,node2-9.grid.orbit-lab.org,node2-10.grid.orbit-lab.org
```

(note that this is all one line, and there are no spaces around the commas).

After the disk image process has finished, we should ensure that all 4 nodes have been successfully imaged and we got a message like the following.

```
-----------------------------
Imaging Process Done
4 nodes successfully imaged
-----------------------------
```

If some nodes are not imaged (e.g. they do not "check in" to the experiment, or they "time out"), repeat the imaging process on just the individual nodes that failed.

Then, turn on your nodes:

```
omf tell -a on -t node2-7.grid.orbit-lab.org,node2-8.grid.orbit-lab.org,node2-9.grid.orbit-lab.org,node2-10.grid.orbit-lab.org
```

After turning the nodes on, wait a few minutes for them to boot. Then, open *four* SSH sessions to your testbed console. In each, SSH in to another one of the four nodes as the root user, e.g. "ssh root@node2-7" in one session, "ssh root@node2-8" in the next, etc.

Then, go on to set up the network.

2.3.1.5 Set up NITOS

At your reserved time, log in to the testbed using your NITOS slice name and keys:

```
ssh -i /PATH/TO/KEY slice_name@nitlab3.inf.uth.gr
```

Identify four nodes that have Atheros 9XXX cards. In these instructions we will use node054,node055,node058,node059 if you are using NITOS. But if one of these nodes is unavailable, you may replace it with another Atheros 9XXX-equipped node - for example, you may use any of the nodes node054-node056, node058-node060, node062-node064.

Once we have logged in to the testbed console and identified four nodes that are available for use in this experiment, we load the `mesh-protocols.ndz` disk image onto the nodes:

```
omf load -i mesh-protocols.ndz -t node054,node055,node058,node059
```

(note that this is all one line, and there are no spaces around the commas).

After the disk image process has finished, we should ensure that all 4 nodes have been successfully imaged and we got a message like the following.

```
----------------------------
Imaging Process Done
4 nodes successfully imaged
----------------------------
```

If some nodes are not imaged (e.g. they do not "check in" to the experiment, or they "time out"), repeat the imaging process on just the individual nodes that failed.

Then, turn on your nodes:

```
omf tell -a on -t node054,node055,node058,node059
```

After turning the nodes on, wait a few minutes for them to boot. Then, open four SSH sessions to your testbed console. In each, SSH in to another one of the four nodes as the root user, e.g. ssh -o "StrictHostKeyChecking no" root@node054 in one session, ssh -o "StrictHostKeyChecking no" root@node055 in the next, etc.

Then, go on to set up the network.

2.3.2 Set up mesh network

Now, we will set up a mesh network with B.A.T.M.A.N.

On each node run:

```
modprobe ath9k
modprobe ath5k
```

to load the wireless driver. Then, run:

```
ifconfig wlan0 up
ifconfig wlan0 0.0.0.0 down
ifconfig wlan0 mtu 1532
iwconfig wlan0 mode ad-hoc essid batman-mesh ap 43:5F:6B:88:A7:CF channel 11
```

to set up a mesh network (at layer 2). You can run

```
iwconfig wlan0
```

to verify layer 2 connectivity. If connected, the output should look like this:

```
wlan0     IEEE 802.11abg  ESSID:"batman-mesh"
          Mode:Ad-Hoc  Frequency:2.462 GHz  Cell: 43:5F:6B:88:A7:CF
```

```
          Tx-Power=27 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
```

Afer we set up a mesh network at Layer 2, we will set up a bridge on each node WLAN interface:

```
modprobe bridge

brctl addbr mesh0
brctl addif mesh0 wlan0
```

The purpose of the bridge is to make it possible to filter MAC addresses in a later step, so that we can force arbitrary topologies on the mesh.

Now, we set up the B.A.T.M.A.N. routing protocol on this mesh [5]. On each node, run:

```
modprobe batman-adv
batctl if add mesh0

sysctl -w net.ipv4.ip_forward=1


# Set up IP address
x=$(ifconfig | awk '/inet addr/{print substr($2,6)}' | head -n 1 | cut -f3 -
d'.')
y=$(ifconfig | awk '/inet addr/{print substr($2,6)}' | head -n 1 | cut -f4 -
d'.')

ifconfig mesh0 up
ifconfig wlan0 up
ifconfig bat0 192.168.$x.$y netmask 255.255.0.0
```

Wait a few minutes for the B.A.T.M.A.N. messages to propagate through the network. Then, you can find out what neighbours each node can "see" by running

```
batctl n
```

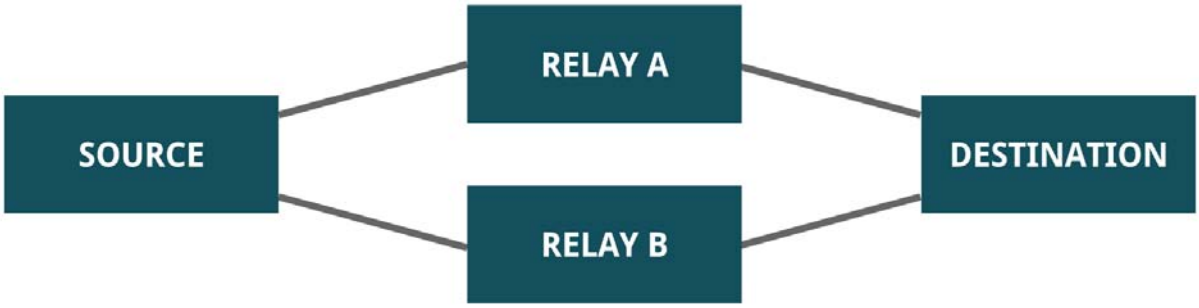
on each node. You should be able to verify that each node can "see" exactly three neighbors, e.g.

```
[B.A.T.M.A.N. adv 2017.0, MainIF/MAC: mesh0/00:60:b3:25:c0:3a
(bat0/de:ec:6a:4f:b9:a7 BATMAN_IV)]
IF             Neighbor         last-seen
mesh0     00:0c:42:64:b1:e6       0.736s
mesh0     00:0c:42:64:b0:28       0.928s
mesh0     00:0c:42:3a:b6:c2       0.016s
```

2.3.3 Force use of a relay

Of the four nodes in your experiment, choose two to be "end nodes" (one "source" and one "destination") and two to be intermediate relays ("relayA" and "relayB"). We will configure the end nodes so that they do not have a direct link between them (we will use `ebtables` to block direct communication). Thus, when end nodes want to communicate, they send their packets through relay nodes and those respectively forward those packets to the final destination. [10]

This topology is shown here:



In our current setup, all nodes are able to communicate directly with each other. We will use `ebtables` to implement the topology of the figure instead.

On the node that you have designated as the "destination", run `ifconfig | grep wlan0` and note its wireless interface MAC address. The output of the command should look something like this:

```
wlan0     Link encap:Ethernet  HWaddr 00:0c:42:64:b1:e6
```

Now, on the designated "source" node, we will apply filtering to block it from receiving any messages (including OGM messages) that were sent from this MAC address. On the "source" node, run:

```
MAC=00:0c:42:64:b1:e6  # use MAC address of destination

ebtables -A FORWARD -s $MAC -j DROP
ebtables -A INPUT   -s $MAC -j DROP
ebtables -A OUTPUT  -s $MAC -j DROP
```

but replace the MAC address in the first line with the MAC address of *your* "destination" node.

Similarly, get the MAC address of the "source" node and, on the destination node, block its messages. On the "destination" node run:

```
MAC=00:0c:42:64:b1:e6  # use MAC address of source node

ebtables -A FORWARD -s $MAC -j DROP
ebtables -A INPUT   -s $MAC -j DROP
ebtables -A OUTPUT  -s $MAC -j DROP
```

At this point you should verify that the path from source to destination goes through one of the relays. On the "source" node, run

```
batctl traceroute DESTIP
```

where the last argument is the IP address of the "destination" node. (Use `ifconfig bat0` on the "destination" node to find out its IP address.) You should see an intermediate hop through a relay node in the traceroute output. (You may have to wait a few moments for the protocol to realize that the link between source and destination is no longer available, and to adjust the routes.)

2.3.4 Introducing failures in the network

In this experiment we want to examine network links and their ranks depending on the BATMAN protocol evaluation. We also want to examine how the ranks and the network routing behaviour is affected when the conditions on those links change. To do this, we monitor the originator table of BATMAN protocol on nodes, over time.

On the "source" node, run the command

```
batctl o
```

to see the originator table of this node. This table helps us understand to which neighbor a packet is forwarded.

Here is an example of an originator table:

```
[B.A.T.M.A.N. adv 2017.0, MainIF/MAC: mesh0/00:60:b3:25:c0:3a
(bat0/36:56:9c:37:ac:79 BATMAN_IV)]
   Originator        last-seen (#/255) Nexthop           [outgoingIF]
   00:0c:42:64:b0:28    0.060s   (206) 00:0c:42:3a:b6:c2 [     mesh0]
 * 00:0c:42:64:b0:28    0.060s   (251) 00:0c:42:64:b0:28 [     mesh0]
   00:0c:42:3a:b6:c2    0.132s   (216) 00:0c:42:64:b0:28 [     mesh0]
 * 00:0c:42:3a:b6:c2    0.132s   (247) 00:0c:42:3a:b6:c2 [     mesh0]
   00:0c:42:64:b1:e6    0.096s   (208) 00:0c:42:3a:b6:c2 [     mesh0]
 * 00:0c:42:64:b1:e6    0.096s   (221) 00:0c:42:64:b0:28 [     mesh0]
```

when the MAC addresses of the nodes in the network are:

source	00:60:b3:25:c0:3a
relayA	00:0c:42:3a:b6:c2
relayB	00:0c:42:64:b0:28
destination	00:0c:42:64:b1:e6

In the first column of the originator table, it shows the MAC address of the originator node. Then, it shows when that node was last seen on the network. Next, it shows the link quality rank (#/255) of the path to that originator through each possible next hop - the MAC address of the next hop is in the fourth column. In the example above, the MAC addresses of the nodes are:

We can see that the route from source node (00:60:b3:25:c0:3a) to destination node (00:0c:42:64:b1:e6), uses relay B (00:0c:42:64:b0:28) as an intermediate hop for its transmission, and the rank of this path is 221. We can also see that routes to 00:0c:42:3a:b6:c2 and 00:0c:42:64:b0:28 (the relays) are direct links and no intermediate hops are used for them. Their respective ranks are 247 and 251.

For a better understanding of BATMAN routing we will examine how these next-hop decisions and the estimated link quality are affected over time for when a link fails or recovers from a failure.

For convenience, we have written a script for real time monitoring of originator table links to destination node, available in this gist. On the source node, we download the script:

```
wget -O relay-ranks-monitoring.sh https://git.io/vHiMN
```

Then run this script it on the source node. (In the command, use the MAC addresses of the destination, relay A, and relay B nodes, respectively. For the last argument, use the filename to which you want to save the results.)

```
bash relay-ranks-monitoring.sh DEST_MAC RELAY_A_MAC RELAY_B_MAC batman-no-failure.csv
```

The script will show the entries (next hops and rank) in the originator table for paths to the destination node, and will update every second. The output of the script will be something like this:

```
  originator(MAC)  rank(hop-A)  rank(hop-B)  time(sec)
00:0c:42:64:b1:e6        217          217         0
00:0c:42:64:b1:e6        217          216         1
00:0c:42:64:b1:e6        217          215         2
00:0c:42:64:b1:e6        217          214         3
00:0c:42:64:b1:e6        217          212         4
...
```

(Press `Ctl+C` to stop the script execution.)

Now we will see the behaviour of the link from source to destination node through Relay A, when the link into Relay A from the end nodes is down. On the source node we run the following command:

```
bash relay-ranks-monitoring.sh DEST_MAC RELAY_A_MAC RELAY_B_MAC batman-relay-a-failure.csv
```

While this is running, on Relay A run:

```
SOURCE=SOURCE_MAC
DESTINATION=DEST_MAC
```

```
ebtables -A FORWARD -s $SOURCE -j DROP
ebtables -A INPUT   -s $SOURCE -j DROP
ebtables -A OUTPUT  -s $SOURCE -j DROP
ebtables -A FORWARD -s $DESTINATION -j DROP
ebtables -A INPUT   -s $DESTINATION -j DROP
ebtables -A OUTPUT  -s $DESTINATION -j DROP
```

where you specify the MAC address of the source and destination nodes in the first two lines.

On the output, we can see that link ranks through Relay A start dropping.

```
 originator(MAC)  rank(hop-A)  rank(hop-B)  time(sec)
00:0c:42:64:b1:e6         192          190          0
00:0c:42:64:b1:e6         191          189          1
00:0c:42:64:b1:e6         191          188          2
00:0c:42:64:b1:e6         191          190          3
00:0c:42:64:b1:e6         190          191          4
...

00:0c:42:64:b1:e6         105          191          33
00:0c:42:64:b1:e6         102          191          34
00:0c:42:64:b1:e6         101          193          35
00:0c:42:64:b1:e6         99           195          36
00:0c:42:64:b1:e6         0            195          37
```

After some time, the ranks of the link facing failures drop to zero. We stop the execution of the script (`ctl+c`) and wait for one minute.

Now, we will see how this links recovers when we stop the failures. On the source node, we execute again the script by giving the commands:

```
bash relay-ranks-monitoring.sh DEST_MAC RELAY_A_MAC RELAY_B_MAC batman-relay-a-recovery.csv
```

On Relay A, we run the command:

```
ebtables -F
```

On source node, we can see the link ranks through Relay A recovering:

```
 originator(MAC)  rank(hop-A)  rank(hop-B)  time(sec)
00:0c:42:64:b1:e6         100          198          0
00:0c:42:64:b1:e6         102          198          1
00:0c:42:64:b1:e6         104          197          2
00:0c:42:64:b1:e6         107          195          3
00:0c:42:64:b1:e6         111          194          4
...
```

```
00:0c:42:64:b1:e6         192         194         36
00:0c:42:64:b1:e6         193         194         37
00:0c:42:64:b1:e6         194         194         38
00:0c:42:64:b1:e6         194         193         39
00:0c:42:64:b1:e6         195         191         40
```

When the Relay A links recover, we press `ctl+c` and stop the script.

Now we repeat the same process for the link ranks of the *Relay B*. First we introduce some link failures. On the source node run:

```
bash relay-ranks-monitoring.sh DEST_MAC RELAY_A_MAC RELAY_B_MAC batman-relay-b-failures.csv
```

Next on Relay B:

```
SOURCE=SOURCE_MAC
DESTINATION=DEST_MAC

ebtables -A FORWARD -s $SOURCE -j DROP
ebtables -A INPUT   -s $SOURCE -j DROP
ebtables -A OUTPUT  -s $SOURCE -j DROP
ebtables -A FORWARD -s $DESTINATION -j DROP
ebtables -A INPUT   -s $DESTINATION -j DROP
ebtables -A OUTPUT  -s $DESTINATION -j DROP
```

We notice on source that ranks of paths through Relay B start dropping.

```
 originator(MAC)  rank(hop-A)  rank(hop-B)  time(sec)
00:0c:42:64:b1:e6         191         196         0
00:0c:42:64:b1:e6         192         196         1
00:0c:42:64:b1:e6         192         196         2
00:0c:42:64:b1:e6         195         195         3
00:0c:42:64:b1:e6         196         194         4
...

00:0c:42:64:b1:e6         198         104         78
00:0c:42:64:b1:e6         198         103         79
00:0c:42:64:b1:e6         198         103         80
00:0c:42:64:b1:e6         198         102         81
00:0c:42:64:b1:e6         198         0           82
```

We stop the script on the source node and start it again after one minute with the commands:

```
bash relay-ranks-monitoring.sh DEST_MAC RELAY_A_MAC RELAY_B_MAC batman-relay-b-recovery.csv
```

Finaly, we run this command on Relay B:

```
ebtables -F
```

We can see the ranks of the paths through Relay B recover:

```
 originator(MAC)  rank(hop-A)  rank(hop-B)  time(sec)
00:60:b3:25:c0:1b         194         102         0
00:60:b3:25:c0:1b         194         102         1
00:60:b3:25:c0:1b         194         103         2
00:60:b3:25:c0:1b         194         104         3
00:60:b3:25:c0:1b         194         104         4
...

00:60:b3:25:c0:1b         194         190         63
00:60:b3:25:c0:1b         195         191         64
00:60:b3:25:c0:1b         193         191         65
00:60:b3:25:c0:1b         193         193         66
00:60:b3:25:c0:1b         193         194         67
```

Four CSV files were saved to the "source" node. You can retrieve these files with SCP or with transfer.sh, and then create a visualization as in the Results section.

3. Emulations creating a specific mesh network topology

In this experiment, we set up a mesh network similar to a part of a live community network, and observe how packets are routed across multiple hops in this network.

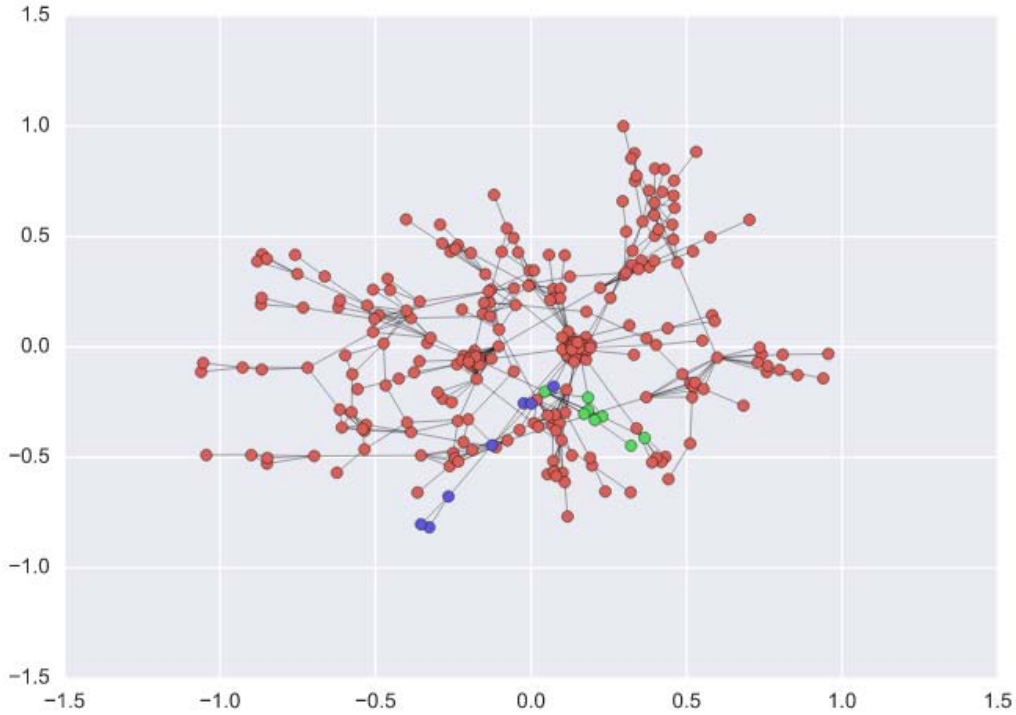
3.1 Background

Community networks are wireless mesh networks that are built from the ground up; as each member joins the network, it becomes a node in the network and may carry other members' traffic. Some members also have conventional Internet connectivity through an ISP, and share their service with the community network.

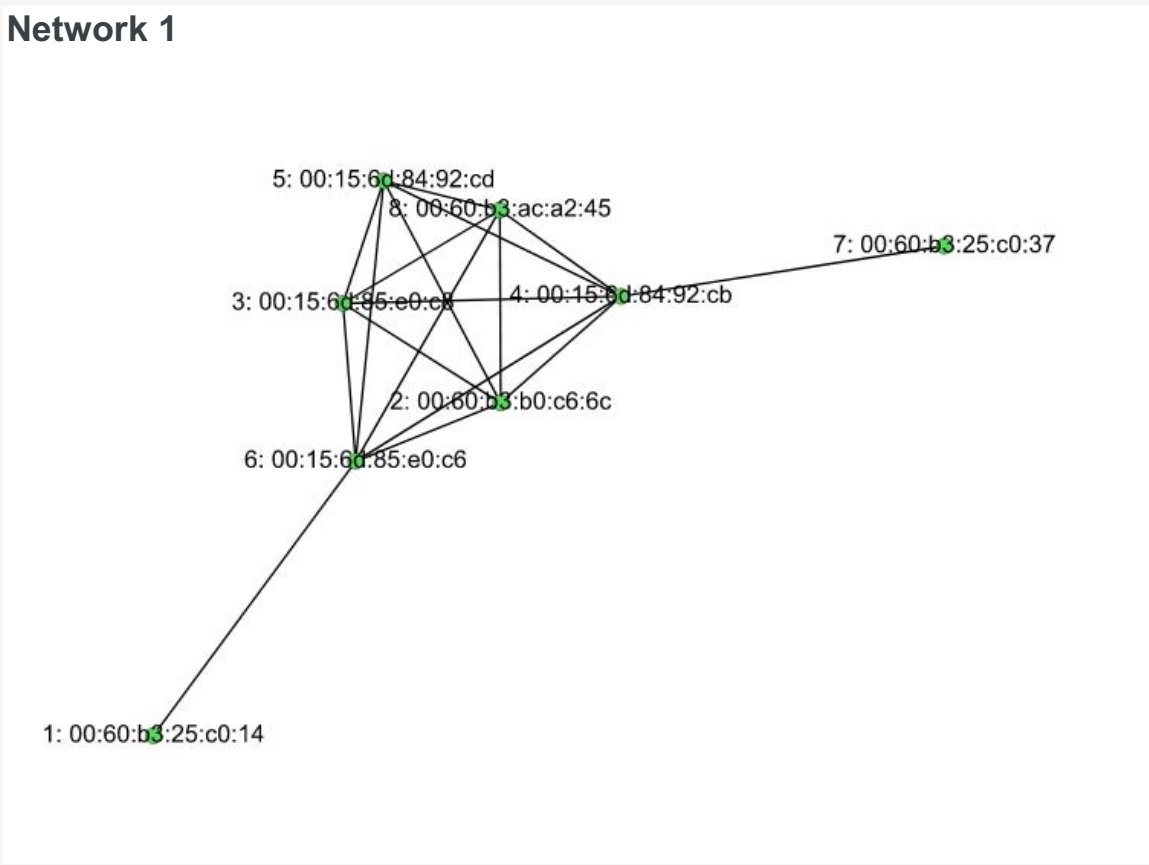
The Representing community network topologies on GENI experiment describes how to set up a topology on GENI that has the same structure as a part of the Funkfeuer Graz community network. However, that experiment runs on wired GENI resources. [9],[11],[12] For some applications, it may be desirable to set up a topology that mimics a part of a community network on a testbed with real wireless links.

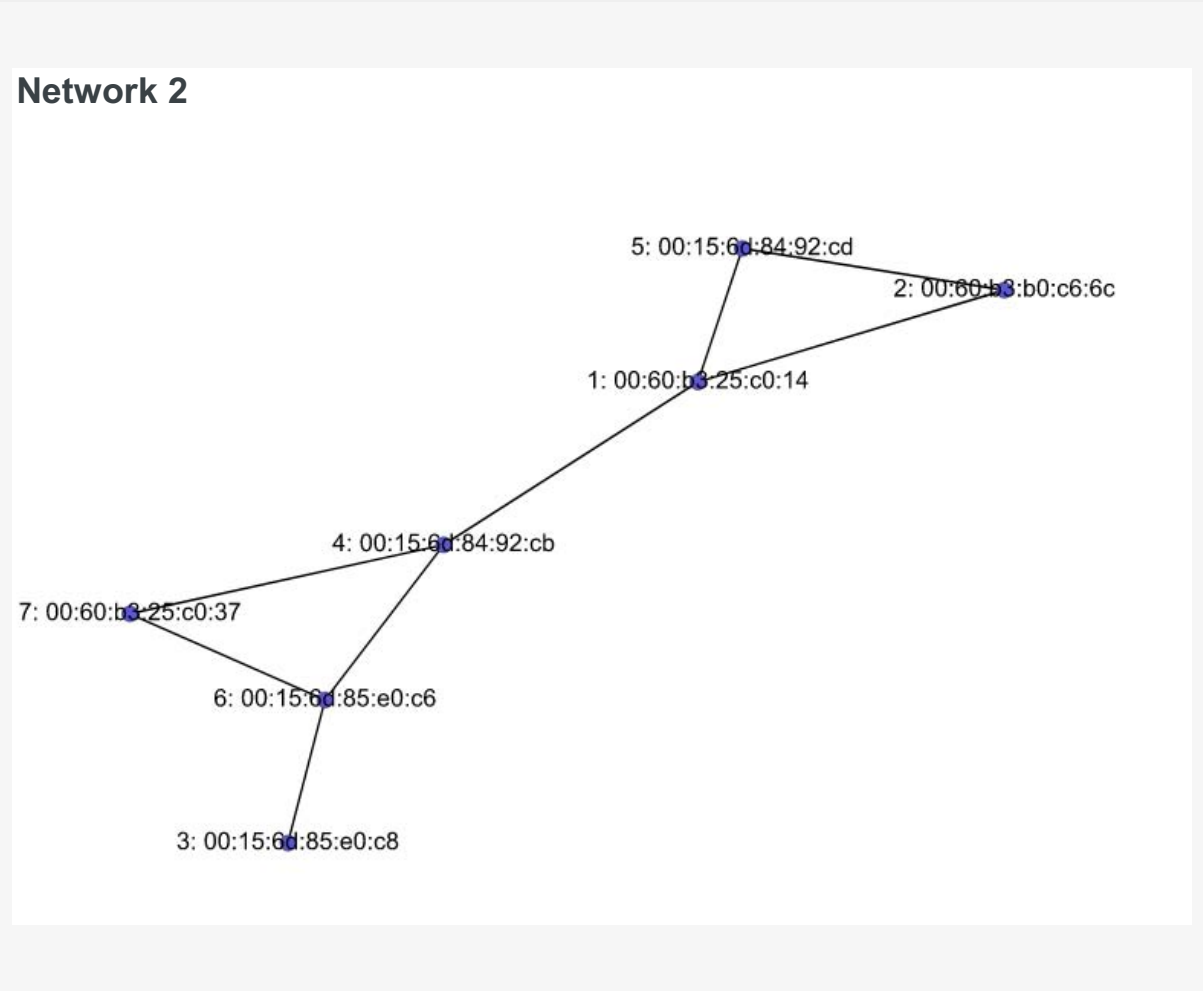
In this experiment, we will show how to do this on the sb4 testbed on ORBIT. This testbed has a programmable attenuation matrix, which we can use to set up any arbitrary wireless topology. This allows us to run experiments involving a small subset (9 nodes or fewer) of a community network. [7]

Using a similar procedure as in Representing community network topologies on GENI, we split off two "subgraphs" in the FF Graz community network that have nine nodes or fewer [11]. Here, we see these smaller groups colored green (network 1) or blue (network 2) within the larger community network:



And here we see each small group on its own, together with its realization on the sb4 testbed (each node in each image is marked with the number of the sb4 testbed node on which it will be realized, and its MAC address):



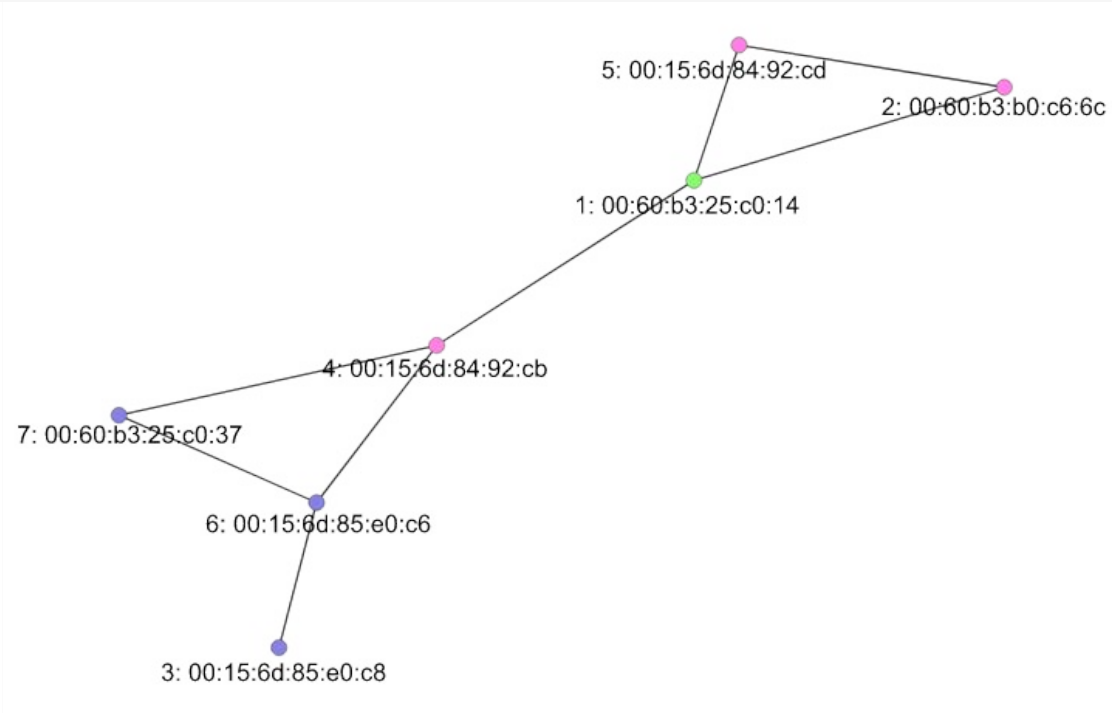


In this experiment, we will set up one of these topologies with real wireless links, using the programmable attenuation matrix on sb4. Then, to validate our setup, we'll observe the behaviour of a mesh routing protocol in the network.

3.2 Results

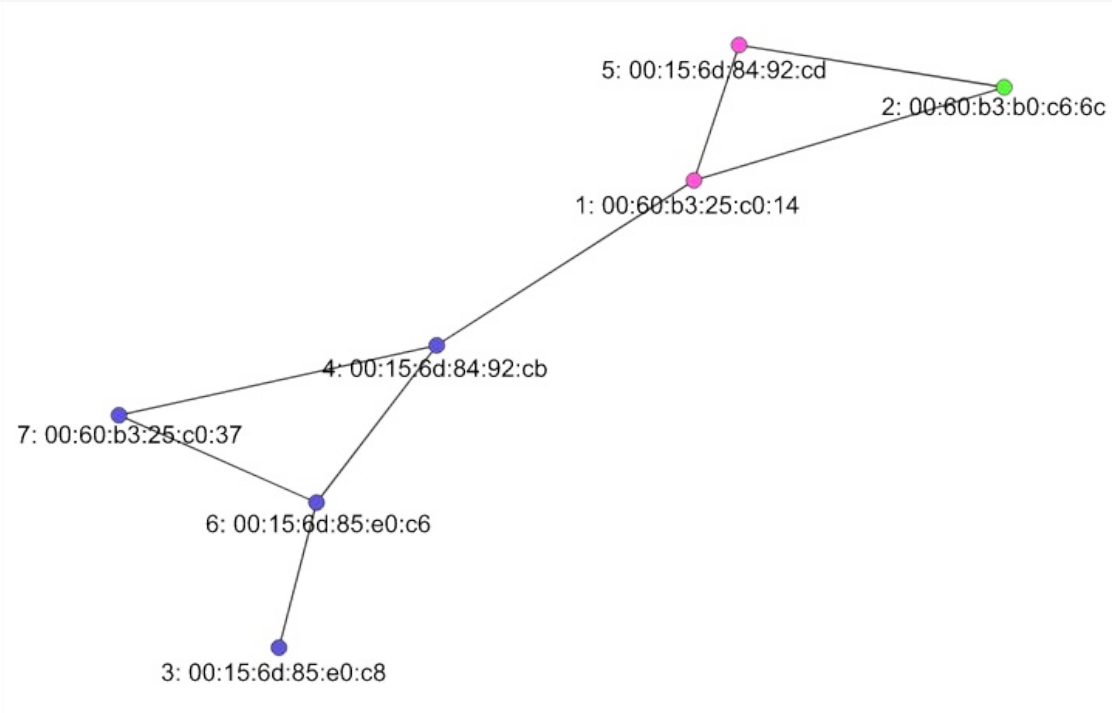
We will use the output of a mesh routing protocol to validate that our network topology on sb4 mimics the community network subgraph that we have selected.

For example, we created the following table to show how we validate our setup for network #2. In the following table, each row represents a different node. On the left, we show the node (marked in green in the figure) and its neighbours (in pink). On the right, we show the neighbour table created by the B.A.T.M.A.N. mesh routing protocol. We can see that the network topology is correctly represented on the testbed:



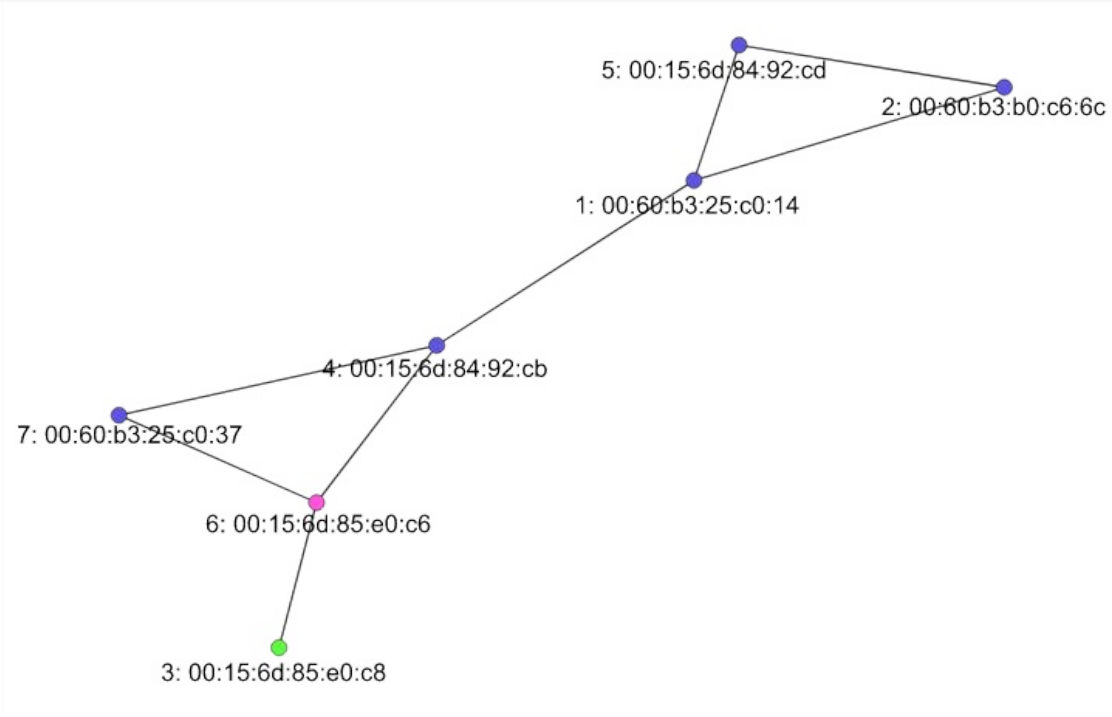
node1-1: 00:60:b3:25:c0:14

IF	Neighbor	last-seen
wlan0	00:15:6d:84:92:cb	0.452s
wlan0	00:15:6d:84:92:cd	0.260s
wlan0	00:60:b3:b0:c6:6c	0.264s



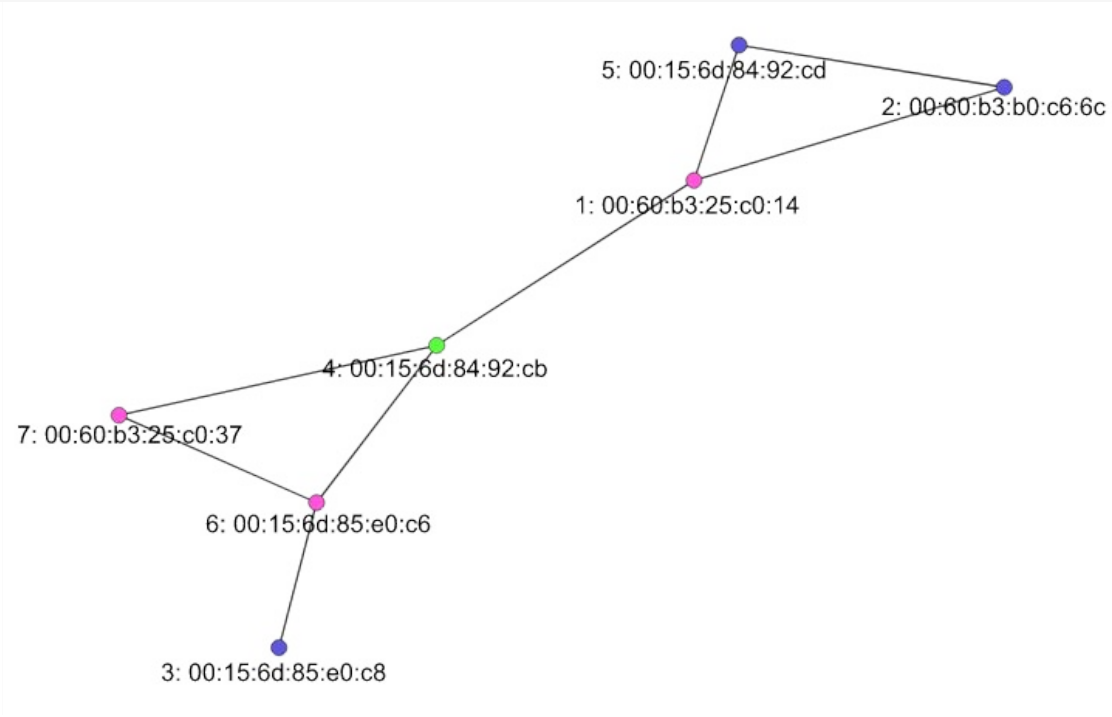
node1-2: 00:60:b3:b0:c6:6c

IF	Neighbor	last-seen
wlan0	00:60:b3:25:c0:14	0.892s
wlan0	00:15:6d:84:92:cd	0.480s



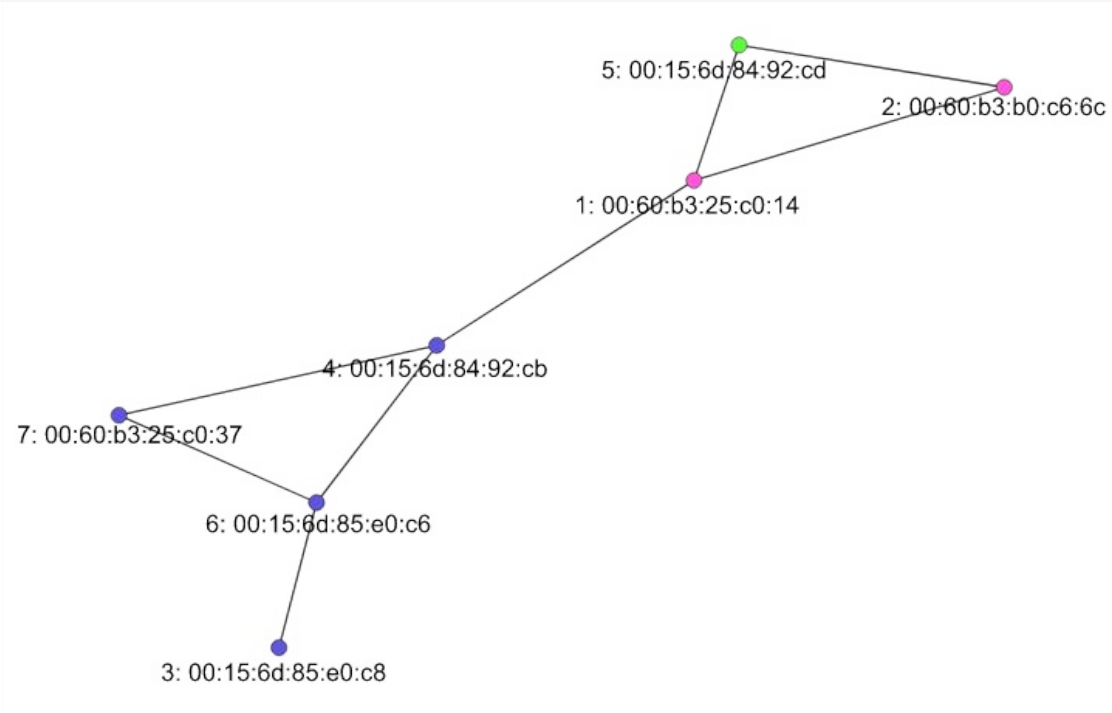
node1-3: 00:15:6d:85:e0:c8

IF	Neighbor	last-seen
wlan0	00:15:6d:85:e0:c6	0.624s



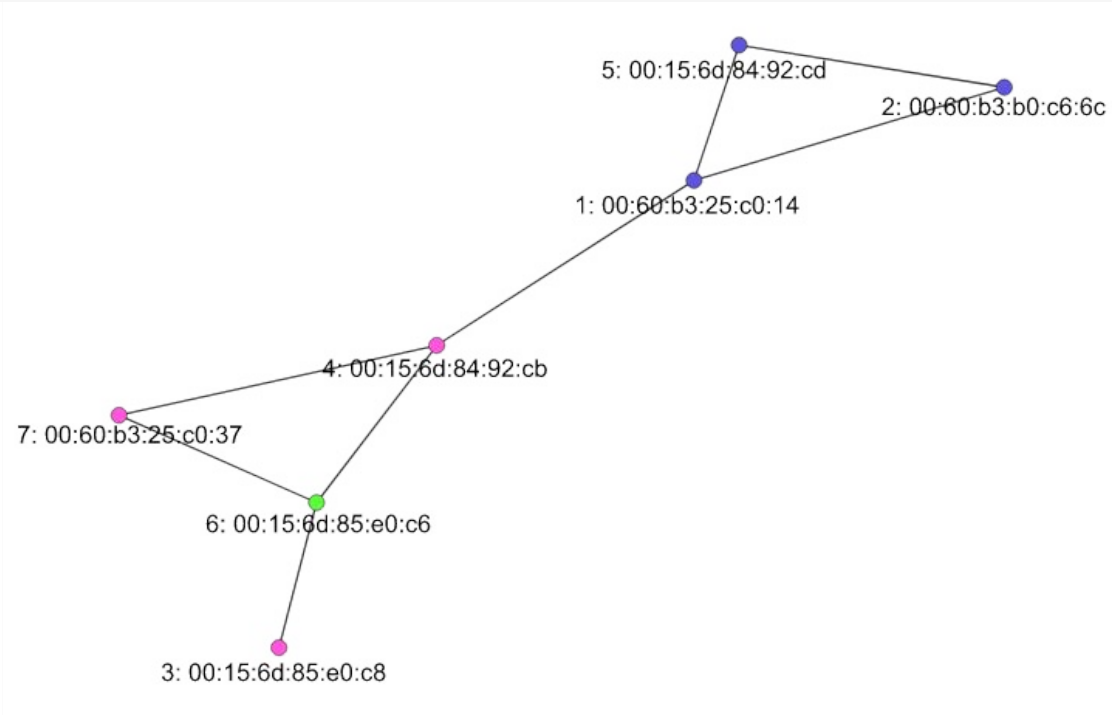
node1-4: 00:15:6d:84:92:cb

IF	Neighbor	last-seen
wlan0	00:60:b3:25:c0:14	0.500s
wlan0	00:60:b3:25:c0:37	0.376s
wlan0	00:15:6d:85:e0:c6	0.008s



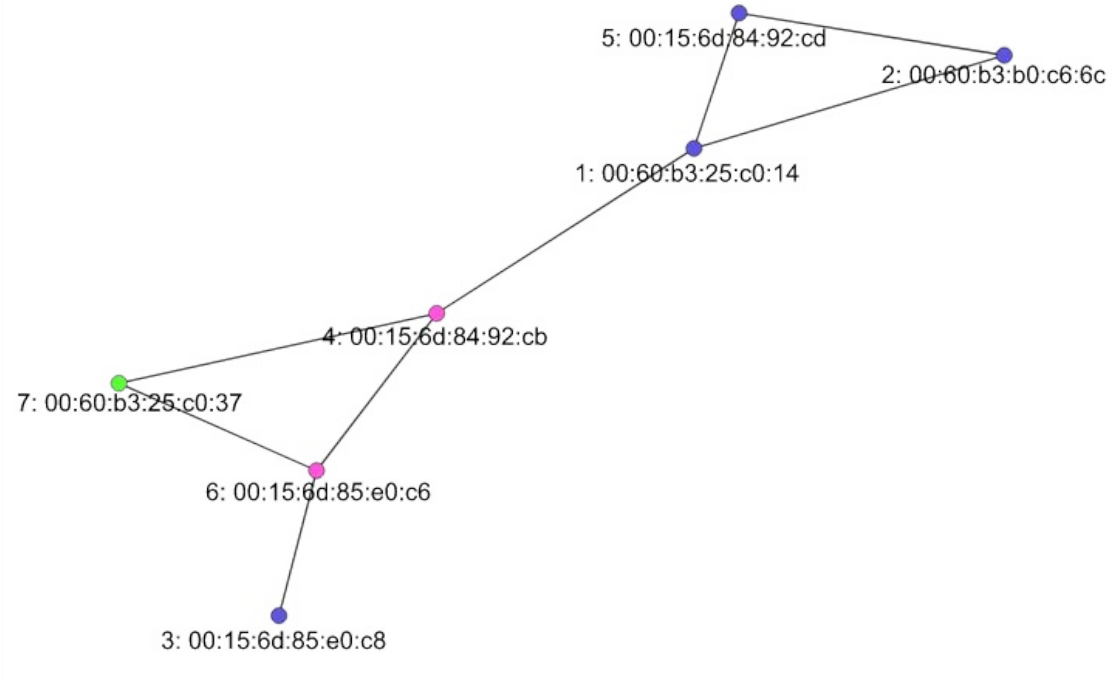
node1-5: 00:15:6d:84:92:cd

IF	Neighbor	last-seen
wlan0	00:60:b3:25:c0:14	0.908s
wlan0	00:60:b3:b0:c6:6c	0.436s



node1-6: 00:15:6d:85:e0:c6

IF	Neighbor	last-seen
wlan0	00:60:b3:25:c0:37	0.840s
wlan0	00:15:6d:84:92:cb	0.688s
wlan0	00:15:6d:85:e0:c8	0.416s



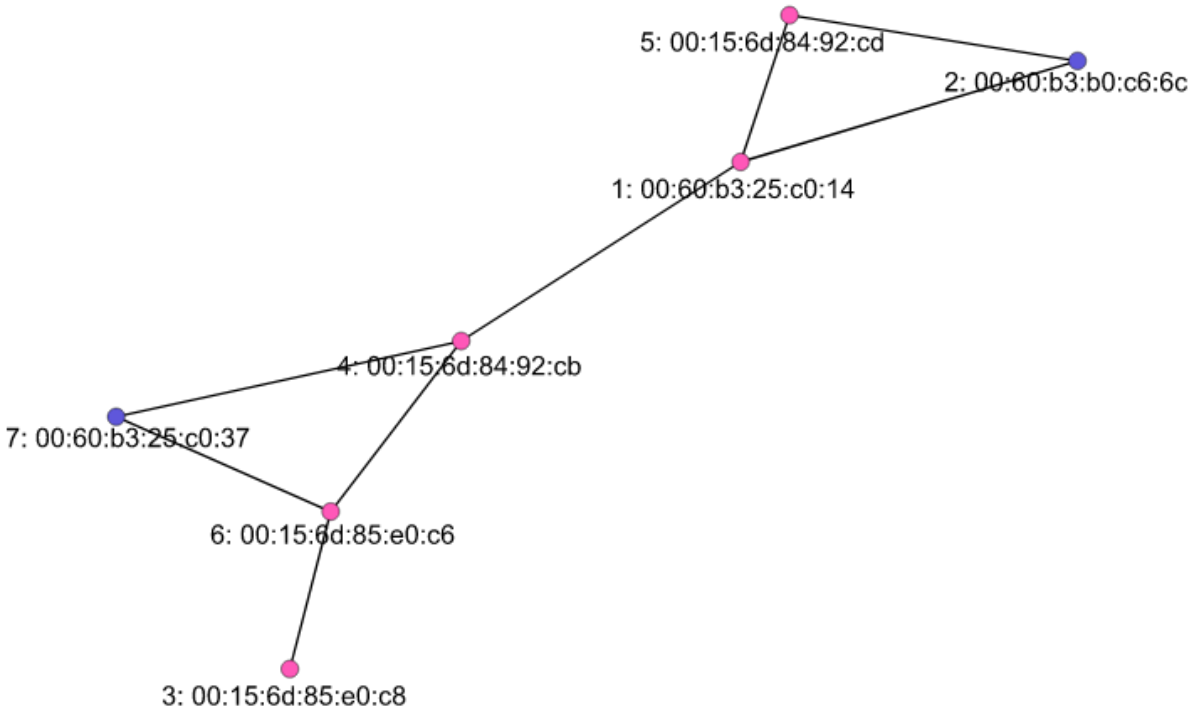

```
node1-7: 00:60:b3:25:c0:37

  IF       Neighbor           last-seen
wlan0    00:15:6d:85:e0:c6    0.964s
wlan0    00:15:6d:84:92:cb    0.160s
```

We can also trace the path a packet takes through the network. For example, we can trace the path from node1-3 to node1-5:

```
root@node1-3:~# batctl traceroute 192.168.1.5
traceroute to 192.168.1.5 (00:15:6d:84:92:cd), 50 hops max, 20 byte packets
 1: 00:15:6d:85:e0:c6  4.172 ms  1.341 ms  1.578 ms
 2: 00:15:6d:84:92:cb  3.476 ms  2.865 ms  3.370 ms
 3: 00:60:b3:25:c0:14  4.147 ms  4.211 ms  4.088 ms
 4: 00:15:6d:84:92:cd  5.418 ms  5.926 ms  5.196 ms
```

and then verify from the graph that this is the expected path (through nodes marked in pink):



3.3 Run my experiment

At your reserved time, SSH into

```
sb4.orbit-lab.org
```

with your GENI wireless username and associated keys.

When you first log in to the "sb4" console, you should reset sb4's programmable attenuation matrix to zero attenuation between all pairs of nodes [7]. From the "sb4" console, run

```
wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/setAll?att=0"


wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/selDevice?switch=1&port=1"
wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/selDevice?switch=2&port=1"
wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/selDevice?switch=3&port=1"
wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/selDevice?switch=4&port=1"
```

3.3.1 Set up the testbed nodes

Then, load the "mesh-protocols" disk image onto the testbed nodes:

```
omf load -i mesh-protocols.ndz -t system:topo:all
```

Wait for this process to finish. If some nodes are not imaged (e.g. they do not "check in" to the experiment, or they "time out"), repeat the imaging process on the individual nodes that failed. For example, if node1-2 and node1-9 did not load the disk image successfully, then run

```
omf load -i mesh-protocols.ndz -t node1-2.sb4.orbit-lab.org,node1-9.sb4.orbit-lab.org
```

It may take a few tries to get the image onto all of the nodes.

After the disk image has been loaded onto all 9 nodes in sb4, turn them on with

```
omf tell -a on -t system:topo:all
```

3.3.2 Set up the community network topology

Next, we will use the programmable attenuation matrix on sb4 to set up one of the community network topologies. We will set 10 dB of attenuation between all "connected" nodes, and 63 dB of attenuation (the maximum) between nodes that are not connected. [7],[13]

To set up network #1, run this gist:

```
wget https://git.io/vHk6K | bash
```

on the sb4 console. Or, to set up network #2, run this gist:

```
wget -qO- https://git.io/vHk6r | bash
```

on the sb4 console.

You can see a visual representation of the current sb4 topology at http://www.orbit-lab.org/status/sb4/network.

3.3.3 Set up a mesh network with B.A.T.M.A.N.

Open up eight terminal sessions (if using network #1) or seven terminal sessions (if using network #2). In each, SSH into the sb4 console, and from there, SSH into each of the nodes as the root user. For example, to log in to node1-1 from the sb4 console run

```
ssh root@node1-1
```

For network #1, log into nodes 1-1 through 1-8, and for network #2 log into nodes 1-1 through 1-7.

Then, on each node, run

```
modprobe ath9k
modprobe ath5k
sleep 1

ifconfig wlan0 up
ifconfig wlan0 0.0.0.0 down
ifconfig wlan0 mtu 1532
iwconfig wlan0 mode ad-hoc essid community-mesh ap 02:B8:C0:08:78:80 channel 11
```

to set up a mesh network (at layer 2). You can run

```
iwconfig wlan0
```

to verify layer 2 connectivity. If connected, the output should look like this:

```
wlan0     IEEE 802.11abg  ESSID:"community-mesh"
          Mode:Ad-Hoc  Frequency:2.462 GHz  Cell: 02:B8:C0:08:78:80
          Tx-Power=27 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
```

Now, we set up the B.A.T.M.A.N. routing protocol on this mesh [5]. On each node, run:

```
modprobe batman-adv
batctl if add wlan0

sysctl -w net.ipv4.ip_forward=1

# Set up IP address
y=$(hostname -s | cut -f2 -d'-')
ifconfig wlan0 up
ifconfig bat0 192.168.1.$y
```

Wait a few minutes for the B.A.T.M.A.N. messages to propagate through the network. Then, you can find out what neighbors each node can "see" by running

```
batctl n
```

on each node. You should be able to verify that each node can "see" exactly the neighbors that it is supposed to, depending on which network we chose to mimic.

You can also trace the route a packet takes through the network by running

```
batctl traceroute DESTINATION
```

on the source node, where in place of "DESTINATION" you give the IP address of the destination node. (The destination IP address will be 192.168.1.X, where X is that node number.) It will show the MAC address of each intermediate hop along the path to the destination.

3.3.4 Building on this experiment: varying link attenuation

With the programmable attenuation matrix, you can set any attenuation you want - so you can degrade some links and see what would happen to the performance of the community network.

To change the attenuation between a pair of nodes, run

```
wget -qO- "http://internal2dmz.orbit-lab.org:5054/instr/set?portA=1&portB=2&att=20"
```

where in place of the bolded values, you specify the node numbers of the two nodes, and the attenuation (in dB, any integer value from 0 to 63) that you want between them.

References

[1] Mesh networking, “https://en.wikipedia.org/wiki/Mesh_networking”

[2] Mesh Networks – P2P Foundation, “http://wiki.p2pfoundation.net/Mesh_Networks”

[3] Wireless Mesh Networks/Mesh network basics, “https://en.wikibooks.org/wiki/Wireless_Mesh_Networks/Mesh_network_basics”

[4] B.A.T.M.A.N. Advanced - Routing Protocol, “https://www.open-mesh.org/”

[5] B.A.T.M.A.N., “https://en.wikipedia.org/wiki/B.A.T.M.A.N.”

[6] WITest Testbed, “https://witestlab.poly.edu/site/”

[7] Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT), “https://geni.orbit-lab.org/”

[8] NITOS Experimental Portal, “http://nitos.inf.uth.gr/”

[9] GENI (Global Environment for Network Innovations), “http://www.geni.net/”

[10] Ebtables firewall tool, “http://ebtables.netfilter.org/”

[11] Representing community network topologies on GENI, “https://witestlab.poly.edu/blog/representing-community-network-topologies-on-geni/”

[12] Funkfeuer Graz Statistics, “https://www.ffgraz.net/”

[13] JFW Industries - RF Transceiver Test System, “https://www.jfwindustries.com/product-category/test-systems/transceiver-testing/”